Introducing Geometric Constraint Expressions into Robot Constrained Motion Specification and Control

Gianni Borghesan[1], Enea Scioni[1,2], Abderrahmane Kheddar[3], Herman Bruyninckx[1,4]

Abstract—
THE problem of robotic task definition and execution was pioneered by Mason, [1], who defined *setpoint constraints* where the position, velocity, and/or forces are expressed in one particular *task frame* for a 6-DOF robot. Later extensions generalized this approach to constraints in *i) multiple* frames, *ii) redundant* robots, *iii) other sensor spaces* such as cameras, and *iv) trajectory* tracking. Our work extends *tasks definition* to *i) expressions* of constraints, with a focus on expressions between geometric entities (distances and angles), in place of *explicit set-point constraints*, *ii)* a systematic *composition* of constraints, *iii) runtime monitoring* of all constraints (that allows for *runtime sequencing* of constraint sets via, for example, a *Finite State Machine*), and *iv) formal* task descriptions, that can be used by *symbolic reasoners* to plan and analyse tasks. This means that tasks are seen as ordered groups of constraints to be achieved by the robot's motion controller, possibly with different set of geometric expressions to measure outputs which are not controlled, but are relevant to assess the task evolution. Those monitored expressions may result in events that trigger switching to another ordered group of constraints to execute and monitor.

For these task specifications, formal language definitions are introduced in the JSON-schema modeling language.

Index Terms—**Software, Middleware and Programming Environments; Motion and Path Planning; Behaviour-Based Systems.**

I. Introduction

Specifying *what* one wants a robot to do, instead of having to code manually *how* it has to do it, is still a challenge for the research community; for example, in Fig. 1, where a simple open-the-drawer task is represented, what the robot *does* is to open the drawer.

Manuscript received: August 12, 2015; Revised October 6, 2015; Accepted November 27, 2015.

This paper was recommended for publication by Editor Tamim Asfour upon evaluation of the Associate Editor and Reviewers' comments. The authors gratefully acknowledge the support from the European Commission's FP7 project "RoboHow" (FP7-ICT-288533), and KU Leuven's *Geconcerteerde Onderzoeks-Actie* "Global real-time optimal control of autonomous robots and mechatronic systems".

[1]Mechanical Engineering, University of Leuven, Belgium.

[2]Engineering Department (ENDIF), University of Ferrara, 44122 Ferrara, Italy.

[3]CNRS-AIST Joint Robotics Laboratory (JRL), UMI3218/RL, Japan, and CNRS-Université Montpellier, LIRMM, Interactive Digital Human, France.

[4]Mechanical Engineering, Eindhoven University of Technology, the Netherlands.

Digital Object Identifier xxxxxxxxxxxxxxxxxxx

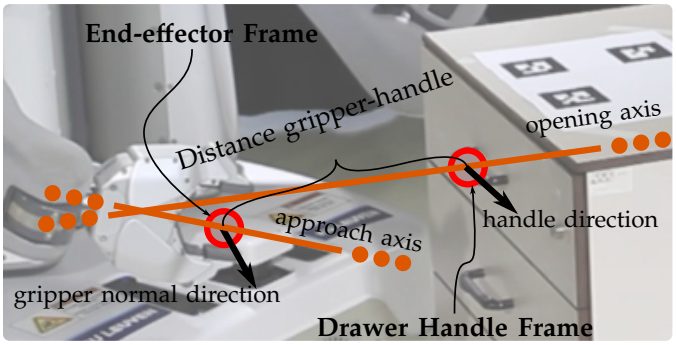


Figure 1. The representation of the a grasping task for the (topmost) handle and open-the-drawer task needs the definition of many geometric features upon which constraints must be enforced.

However, the task specification concerns the satisfaction of some geometric constraint (distance to zero, correct alignment, *etc.*), in order to grasp the uppermost handle. The *how*-to-do task specification approach can be made easy if regarded from the perspective of recent *constrained optimization* based control. Instantaneous motion of the robot's joints is computed as the solution of a constrained optimization problem in which:

- the *objective function* defines the general behavior (e.g. time, energy consumption...) and can even include weight-prioritized tasks and/or utility functions;
- the *constraints* expressed as *equality* as well as *inequality* functions, define the sub-space in which the solution belongs to in terms of the task and robot "state variables".

Moreover, the *switching* between different sets of objective functions and constraints is ruled on the basis of monitoring, the extent to which some functions (not necessary constraint functions) assume given values during the task execution. Previous work such as, *e.g.*, *Stack of Tasks* ("SoT", [2]), *instantaneous Task Specification and Control* ("iTaSC", [3]), *Whole Body Control* ("WBC", [4]), or a revised versions of the *Task Frame Formalism*, ("TFF", [5], [6]), are examples of such a trend.

However, a large part of these formalisms, a robot can be driven to achieve complex tasks that require the definition of many constraints in different spaces and in different reference frames. However, large part of the research has focused on maturing the control aspects

(namely the numerical solver part), leaving the way tasks are designed and formalized untreated.

Summing up, the state of the practice is that:

- current frameworks do not follow the rule of the *separation of concerns* (see *e.g.* [7]): they do not separate the configuration of the hardware platform, of the data communication and synchronization between software components, *etc.*, from the task definition.
- a formal semantics is still missing, reducing the opportunities to use task planning at the symbolic level to automatically generate tasks.

Looking at the above considerations' state-of-the-art, we are convinced that a fundamental step to ease the employment of constraint-based systems and task-based programming is to separate the task description from its implementation and describe the first with a *Domain Specific Language* that follows semantic rules. Our main contribution is therefore the design of a DSL that allows abstraction of: *i)* the space where tasks are expressed, which is defined in terms of a symbolic *expressions*, and *ii)* the type of control to be enforced on such space, which is defined in terms of the achieved *behaviour*.

II. Tasks as sets of constraints: fundamentals

Our goal is to describe a task constraint with a *minimal set* (of properties, relations, equations...), in such a way that the *specification* is abstracted (as much as possible) by the *implementation* of how *to solve* for the constraint during the *execution* of the task. We start considering the simplest task description formalism, the *Task Frame Formalism*, [1], [5], where each constraint is defined employing a minimal set of specifications, namely: *i)* the controlled and task frames, *ii)* selection matrix $ that expresses which, among the six components of the Cartesian space, are controlled in position and which are controlled in force, and *iii)* the reference values. Thus, the description of a task (in the Task Frame Formalism) consists of at most six non-conflicting constraints, that are applied in the controlled frame (that normally corresponds to the robot end effector) and expressed in a frame where position and orientation are chosen to represents elements of interest for the task itself. Employing the Task Frame Formalism approach, Bruyninckx and De Schutter, [5], already proposed a generic way of defining a tasks as a set of constraints (an example is given in Listing 1).

```
move compliantly {
  with task frame directions
  xt: velocity 0 mm/sec
  yt: velocity 0 mm/sec
  zt: velocity v_des mm/sec
  axt: velocity 0 rad/sec
  ayt: velocity 0 rad/sec
  azt: velocity 0 rad/sec
} until zt force <- f_max N
```

Listing 1. Example of a guarded-motion task definition from [5].

Other works followed this line, either focusing on the specification (*e.g.* [8]), or on the flexibility of constraint definition (*e.g.* [3]).

The Task Frame Formalism separates the task description from its implementation: indeed, Listing 1 fully defines a task, with no explicit dependancies to *robot information*, such as: *i)* the kinematics (redundancy as well as mechanical and actuator constraints), *ii)* the control laws employed in the motion or force control, and *iii)* the perception capabilities (estimation, observation...). Similar considerations can be applied to more recent frameworks (*e.g.* SoT, [2], and WBF, [4]), that, while posing the emphasis on different aspects, maintain the same approach in task definition.

Our extensions to this task description approach are:

- a wider choice of the constrained variables, which, instead of being related to the Cartesian space representation (not always suited to easily describe a generic relationship between two bodies), will be represented as (one of a enumerable set of functions between) *geometric primitives*. In this regard, Sec. IV focuses on the description of the *geometric expressions*, and on the geometric primitives used by these geometric expressions;
- the substitution of the type of *control* (*e.g.*, force and position) with the type of *behaviour* that we want to achieve; which includes the description of the type of control *and* the type of constraint, Sec. V;
- the specification of *measurement expressions* that are employed *to monitor* the task execution state and to trigger changes in the robot behaviour.

The combination of the previous elements (*expressions*, *behaviours* and *monitors*) formally defines a *Domain Specific Language* (DSL) for robotic tasks. Task instances, as well as the specification itself, have been encoded as JavaScript Object Notation (JSON) [9] and JSON-Schema [10] documents, respectively. This allows us to decouple the language-dependent task grounding from the formal description, not only to guarantee portability and legibility, but also to exploit existing tools and model validators. The validation of the model semantic is the first step to allow automatic reasoning systems that generate symbolic plans, [11], [12], or symbolic task scheduling, [13]. Further numerical post-check operations have been implemented in the host language of the controller. For sake of brevity, we will report snippets of the models written in JSON-Schema only for the model point and the entity[1].

III. Analogies with symbolic planning

While we develop this formalism as a way to describe tasks at an abstract level, we later realized that this effort has overlapping goals with works that investigate how to bridge the gap between symbolic planning and continuous action planning. In the latter, actions are often the result of the execution of pre-computed joint trajectories; these trajectories are in turn generated from one or more target configurations that are then checked

[1]All the models described can be retrieved at https://github.com/gborghesan/jgeom_constr and accompanying material.

Table I
SUMMARY OF GEOMETRIC PRIMITIVES, GROUNDED IN FRAME $\{w\}$.

Geometric primitive:	*symbol*:	*entity composed by*:
point expr. in $\{w\}$	$\boldsymbol{p}_{\{w\}}$	scalars x, y, z
versor expr. in $\{w\}$	$\overline{\boldsymbol{n}}_{\{w\}}$	scalars x, y, z s.t. $\|\cdot\| = 1$
line expr. in $\{w\}$	$\overline{\boldsymbol{n}}_{\{w\}}$	point *origin*, versor *direction*
plane expr. in $\{w\}$	$\mathfrak{P}_{\{w\}}$	point *origin*, versor *normal*

for feasibility and reachability (*e.g.* in [14], [15] RRT-based methods are used). In [16], object geometry is leveraged to reduce computation time of RTT, while in [17] the Task Space Regions are described: TSRs allow to combine constraints expressed in Cartesian space and compute joint trajectories by means of a RRT-based algorithm. Generally, path planning is solved in joint space: as a consequence, if the state of the environment changes, replanning is preferred to active control (*e.g.* [15]), and it is not easy to deal with under-constrained goals. Moreover, planning deals with joint trajectories, and interaction tasks cannot be detailed.

We favour (constraint-based) control rather than full-planning. Constraint-based control is known to be compliant to changes in the environment (as shown in [3]) and allows for the exploitation of unconstrained degrees of freedom (dof). It is naturally suited for hybrid (force-position) task specification. On the other hand, complex environments can cause a task-based control to fall in local minima that a global trajectory planner can avoid. From these considerations, the reader can see that the two approaches are in fact complementary and cover different requirements.

IV. FROM GEOMETRIC ENTITIES TO EXPRESSIONS

This section introduces the geometric concepts of *i) geometric entities* (a geometrical feature such as a point, a versor, a line, a plane, *etc.*), *ii) geometric primitives* (entities expressed in a frame), and *iii) geometric expressions* (a scalar function that relates primitive). Expressions can be non-geometric as well (*e.g.* joint expressions).

Expressions are a relationship that maps *joint-space* values and measurements, to *task-space* values (controlled, measured, *etc.*). The use of expressions is two-fold: *i)* to compute "deviations" from (task-space) positions, or *ii)* to compute the "Jacobian" (the partial derivative of the expression w.r.t. joint angles) that relates joint velocities (forces) to generalized velocities (torques).

Since our focus is *task specification*, no assumption is made on how values and Jacobian are computed (*e.g.* analytically, numerically, with solution of implicit equations, etc.).

A. Geometric entities

The elementary *entities* considered are the *point* and the *versor*; both have *three parameters*, that can be represented by triples (x, y, z). Additionally, the versor must comply with unitary norm constraint. Combinations of points and versors yield the other two common geometric entities, *line* and *plane*, each with *five free parameters* in its representation. (Choosing a *non-minimal* set of representation parameters often simplifies specification.) Since a line is oriented and has an origin, *projection relationships* can be defined (see below).

```
{ "id": "http://.../point#",
  "$schema": "http://json-schema.org/draft-04/↩
      schema#",
  "description": "Point Entity",
  "type": "object",
  "properties": {
    "x": {
      "type": "number",
      "description": "coordinate along x-axis"},
    "y": {
      "type": "number",
      "description": "coordinate along y-axis"},
    "z": {
      "type": "number",
      "description": "coordinate along z-axis"},
    "type": { "enum": ["point"] }
  },
  "required": [ "x", "y", "z", "type" ],
  "additionalProperties": false
}
```

Listing 2. Formal JSON meta-model of a *point entity*.

```
{ "id": "http://.../primitive#",
  "$schema": "http://json-schema.org/draft-04/↩
      schema#",
  "description": "Geometric Primitive",
  "type": "object",
  "properties": {
    "object_frame": {
      "type": "string",
      "description": "reference frame"},
    "entity": { "$ref": "#/definitions/entity" }
  },
  "required": ["object_frame", "entity"],
  "additionalProperties": false,
  "definitions": {
    "entity": {
        "oneOf": [
          { "$ref": "http://.../point#" },
          { "$ref": "http://.../versor#" },
          { "$ref": "http://.../plane#" },
          { "$ref": "http://.../line#" } ]
} } }
```

Listing 3. Formal JSON meta-model of a *geometric primitive*.

B. Geometric primitives

Entities need a *frame* to ground their *coordinate representations*: so, a *geometric primitive* (Listing 3) associates a *geometric entity* with a *frame*. The four possible geometric primitives are reported in Table I, along with the associated mathematical symbol. Formal *meta models* of *point entity* and *primitive* are given in Listing 2–Listing 3; a meta model represents all the *formal constraints* that every individual instance must conform to.

C. Geometric expressions

The list of primitives in Table I is not exhaustive, but suffices for most scalar expressions that describe posi-

Table II
SUMMARY OF GEOMETRIC EXPRESSIONS BETWEEN PRIMITIVES.

	point	line
point	point-point distance	
line	line-point distance projection of point on line	distance btw lines projection (p1-f1) projection (p2-f2)
plane	point-plane distance	

(a) Geometric expressions on distances.

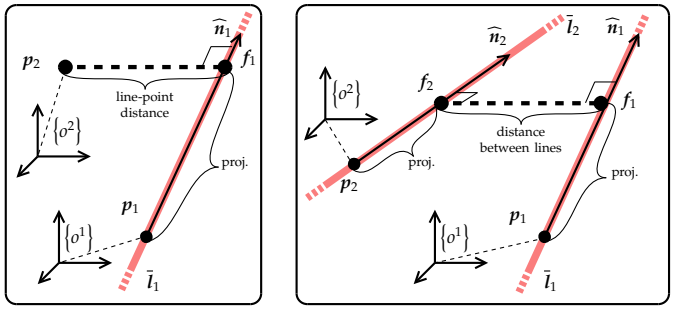
	versor	plane
versor	angle btw versors	
plane	incident angle	angle btw planes

(b) Geometric expressions on angles.

tioning between pairs of objects. For example, Tables IIa and IIb give *distance* and *angle* expressions. Since most of the table entries are self-explaining, we focus only on the explanation of the *line-point* and *line-line distances*.

The *line-point* entry (Fig. 2a) has two expressions: the *line-point distance*, that is the distance between the point p_2 and the point of shortest distance f_1, and the *projection of point on line*, that is the (signed) distance between the points p_1 and f_1.

The *line-line* entry (Fig. 2b) has three distance expressions: the *distance from lines*, distance d between the two lines, the signed distance between the point p_1 and the minimum distance point f_1, and the signed distance between the origin of the line p_1 and the minimum distance point f_2.



(a) Line 1 is expressed in $\{o^1\}$, Point 2 in $\{o^2\}$.

(b) Lines 1 and 2 are expressed in $\{o^1\}$ and $\{o^2\}$, respectively.

Figure 2. Graphical representations of the five possible relations between a point and a line (Fig. 2a), and between two lines (Fig. 2b).

D. Expressions in joint space

Many cases need to express constraints in a robot's joint space, the most obvious cases being limits in torque, position, or velocity.

E. Composite expressions

Many tasks require more complex expressions, either as a *combination* of the elementary expressions above, or based on more complex geometrical *shapes*, or specified directly as geometric "curves". The approach introduced in this paper is *to enumerate* all models that one requires in a specific task context, and with this approach it is straightforward to extend the enumeration list. Here are a number of such higher complexity expressions.

a) Non-scalar expressions: multi-dimensional expressions, as (3D) rotations. Rotations are often used when full orientation is constrained. But in many practical cases, analogous results can be achieved using three *angle between versors* expressions.

b) Sensor-space expressions: in some applications constraints are directly expressed in a sensor space, *e.g.* in visual servoing with eye-in-hand camera. If this kind of relation cannot be expressed by means of geometric expression, a new expression must be formalized, and the underling implementation realized:

$$y = f(\chi_u, q, \dots),$$

where χ_u is the relative position of the measured object w.r.t. the camera, y the measured output, and q the robot configuration.

c) Virtual Fixtures and Mechanisms: are geometric constraints *modelled* as kinematic chains. These computer generated geometric features were pioneered by [18] to overlay real sensory feedback in telerobotics. Later on [19] extended virtual fixtures to generate control primitives in robot teleprogramming of remote robots in the presence of delay. Recently, they are largely exploited in teleoperation and shared control scenarios, especially in the robotic medical field (see [20] and cited works). In this case, more complex primitives are employed, as often the goal of the task is to constrain the end effector to move in a sub-space defined on top of generic curves, and leaving few free directions of motion. These motion primitives are simple and their combination for complex tasks is not always easy or even possible. The concept of *virtual mechanism* proposed in [21] and also in [22] allows a more systematic description of the task and embed a task controller with desired properties (*e.g.* passivity).

Virtual fixtures and mechanisms that rely on linear, rotational, or spherical degrees of freedoms models can be achieved with geometric expressions. More complex scenarios (*e.g.* moving on a generic curve or surface) require an extension to the entity and primitive sets, with new expression models.

V. THE BEHAVIOUR

The "output space" of a task specification is described by *geometric* expressions, but what kind of constraints should be realised in such space must still be specified.

The latter specification is what we call the *behaviour*; a behaviour is a way to specify *what* we want to achieve in a given space while delegating the determination of the control law to the underling solver.

The paper considers the following behaviours:

- **Positioning**, used in position regulation problems.

Table III
LIST OF BEHAVIOURS: EACH BEHAVIOUR IS RELATED TO THE TYPE OF CONTROL AND ITS SPECIFICATION, THE TYPE OF SET-POINT, THE NEEDED MEASUREMENTS (POSITION MEASUREMENT IS ALWAYS NEEDED), AND THE RELATED CONSTRAINT (EITHER EQUALITY OR INEQUALITY). INEQUALITIES NEEDS TWO SET-POINTS, REPRESENTING THE UPPER AND LOWER BOUND VALUES. SPECIFICATION IS OPTIONAL FOR LIMITING BEHAVIOURS.

behaviour	*needs*:								*specification*
	controller	setpoints (one of)					Force measurement	constraints	
		y_d	$\dot{y}_d$	y_d, $\dot{y}_d$	λ_d	λ_d, $\dot{\lambda}_d$		=	<	
Positioning	Position	✓		✓				✓		dominant pole [1/s]
Move	Velocity		✓					✓		dominant pole [1/s]
Physical interaction	Force				✓	✓	✓	✓		dominant pole [1/s]
Compliant motion	Impedance	✓					✓	✓		Stiffness [N/m] or [Nm/rad]
Position Limit	Position	✓×2							✓	*(dominant pole* [1/s]*)*
Velocity Limit	Velocity		✓×2						✓	*(dominant pole* [1/s]*)*
Force limit	Force				✓×2		✓		✓	*(dominant pole* [1/s]*)*

- **Move**, to specify direction and rate of motion, rather than only the desired final position.
- **Physical Interaction**, to control force or impedance occurring in physical contact.
- **Compliant positioning**, to control the position of the system, while allowing for physical compliance in order to cope with unexpected or partially modelled contacts.
- **Limits**, to reduce the space of feasible solutions, preventing the robot from going in undesired positions or to exert excessive forces.

Table III matches behaviours with the needed characteristics of the system, in terms of: *i)* type of control, *ii)* type of set-point (position, velocity, forces, either constant or provided by a trajectory generator), *iii)* and type of constraint. These functionalities (to be provided by the system developers) are introduced here to examplify the meaning of each behaviour. In particular: *i)* in the first three cases, the goal is to have a zero steady state error in either position, velocity, or force, *ii)* in the compliance mode we want to regulate the position, but allowing deviations proportional to force, and *iii)* with limits, we want to specify the bounds. In addition, the first four behaviours need a parameter to indicate *i)* the time constant of the error, in the first three cases, or *ii)* the relation between angular or linear displacement (along the output direction) w.r.t. the disturbance (force or torques respectively).

With the first three laws, we seek an asymptotically zero converging error in position, velocity, or force, respectively, and the error dynamic should be "similar" to a first order system. In the case of position control:

$$\dot{y}_d^\circ = \boldsymbol{K}_p(y_d - y), \tag{1}$$

where $\dot{y}_d^\circ$ is the velocity actuated (see [3]), and the gain $\boldsymbol{K}_p$ is the *specification*, and its dimension is [1/s] regardless of the constrained variable; a feed-forward term ($\dot{y}_d$ or $\dot{\lambda}_d$, in Table III) could also be taken into account.

The compliant motion behaviour, instead, achieves a given displacement from a rest position as response to disturbances (*e.g.* an external force). Henceforth, the tuning parameter represents a desired apparent stiffness:

$$\delta f = \mathbf{K}\delta y. \tag{2}$$

Lastly, in case of limiting-type behaviours, parameters are considered optional, since in many cases (*e.g.* joint position or effort limits), the desired interval should not be violated (and no convergence rule toward limits is needed). On the contrary, in cases where limiting behaviours are used to define "soft tasks" (*e.g.* an auxiliary task as described in Sec. VI-A), these gains can be used to assess a smooth convergence toward the limits following the control equation (2).

The described behaviours cover many of the tasks proposed in literature; however, if the need arises, it is possible to extend the enumerated list with additional behaviours (or add additional parameters to the proposed ones), provided that controllers exist that are able to execute them.

VI. CONSTRAINTS, MONITORS, AND TASK DEFINITIONS

A. Constraint definition

A constraint is defined as a set including the following combination: *i)* an expression, *ii)* a behaviour, and (optionally) *iii)* a trajectory generator, that produces the desired setpoint, as described in Table III.

In all but the simplest tasks, several constraints are enforced together. Since the initial conditions, the environment, and other aspects can be unknown at the time of defining the application, or vary between executions, tasks could results in conflicting objectives that cannot be achieved simultaneously.

For this reason, it is necessary to express explicitly in which way conflicts should be handled at run-time. Designing the behaviour of the system when constraints are conflicting cannot be done in an exhaustive way without peering to the underlying implementation and the possible options that it provides. To the best of the authors' knowledge, two methods are used: *i)* weighting of "deflection" of constraints from their nominal values, employed in velocity- and acceleration- resolved schemes, and *ii)* prioritization of constraints, achieved mostly with null space projectors.

In our experience, we found out that three priority levels suffice, in most applications (see [13], [23] for use cases). We named these three levels as follows:

1) Safety constraints: constraints that are necessary for the *robot platform* or critical surroundings integrity, (*e.g.* hardware limitations, non-desired collision avoidance, sustain balance in humanoid robots, etc.).
2) Primary constraints: the actual constraints to be executed.
3) Auxiliary constraints: constraints that facilitate the execution of primary constraints; an example is the optimization of the robot pose configuration with respect to the manipulability index, [24], another one is gazing, etc.

B. Monitor definition

Driven by the necessity to change the system behaviour once a task is fulfilled, we include monitors in the task description. Monitors observe *expression* variables, raising an event once a particular logical condition is fulfilled on the monitored expression value. Monitors can be used to verify pre-, per- as well as post-conditions, *e.g.* [13]. A monitor can either observe:

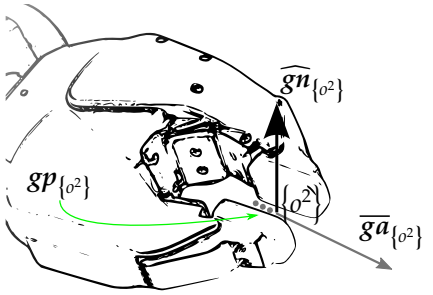
1) a controllable variable (for example a variable that is used as constraint), or
2) a measured quantity that is influenced by the robot actions, but cannot be directly controlled. This quantity can be measured in terms of: *i)* a variable that lives in the space described by an expression, or *ii)* an external monitored value.

For the cases 1) and 2*i*, a monitor is defined as: *i)* an expression (that specifies the space where the variable lives), *ii)* an event name (*e.g.* `finished, failed`), *iii)* the (monitored) variable type (`POSITION`, `VELOCITY`, `FORCE`), that is expressed in the space defined by the expression, *iv)* a comparison type ($<$, $>$, $\in$, $\notin$), *v)* reference value(s). The external monitor (case 2*ii*) allows for composability with external sources, for example, time-out or safety events.

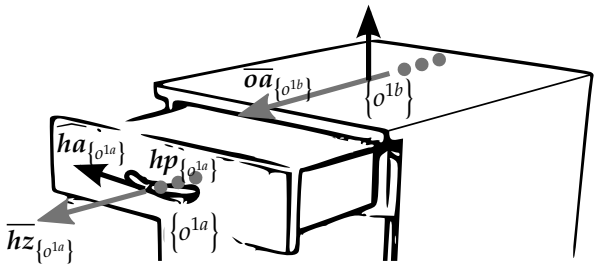
C. Task definition

Finally, the formal definition of a Task wraps the previous things together; it includes *i) at least one* primary constraint, and *ii) at least* one (end-of-task) monitor. *Optionally*, it can have *iii)* safety constraints, and *iv)* auxiliary constraints.

Thus, the minimum specification of the task corresponds to a constraint (an expression, a behaviour that rules which kind of control must be used, and a reference value) and one monitor (that dictates the end of the task execution). Having a well defined end-of-task monitor is instrumental to define the post-condition(s) of the task to be used in plan reasoning and scheduling algorithm (see [13] and references).



(a) Gripper: the line $\overline{ga}_{\{o^2\}}$ represents the direction of approach of grasping, while versor $\widehat{gn}_{\{o^2\}}$ must be parallel to the handle axis direction $\widehat{ha}_{\{o^{1a}\}}$.



(b) Drawer: axis of the handle and direction of opening $\overline{oa}_{\{o^{1b}\}}$ are the two main features.

Figure 3. Some of the geometric entities involved in the task definition.

VII. Task specification example

This section illustrates the proposed approach by means of an open-a-drawer operation composed of the following tasks: *i) Approach the handle of the drawer*, *ii) Grasp the handle*, and *iii) Open the drawer*. In order to show the results of task specification (without introducing the disturbances of controllers, uncertainties, *etc.*), we report the execution of this example run with kinematic simulation, using the software described in [25].

In addition, the accompanying multimedia shows the same experiment executed by the real robot, as well as a "spreading tomato" example, that makes use of force constraints and monitors.

In the following, we briefly describe the tasks in a narrative way[2]. The simulation consist of 3 tasks, whose switching is managed by a finite state machine that reacts upon the event generated by the monitors.

A. Geometric Primitives

The object frames that are involved in the Geometric Primitives are the following:

- $\{o^{1a}\}$, attached to the handle center, with the z-axis along the handle itself.
- $\{o^{1b}\}$, attached to the chest of drawers (fixed in the world).
- $\{o^2\}$, the grasp frame of the robotic hand, *i.e.* the frame that is the center of the grasp once the hand closes.

[2]Specifications are available in the `examples/app` folder of accompanying material.

On such frames, the following *geometric primitives* are defined:

- $\boldsymbol{hp}_{\{o^{1a}\}}$ (`handle_position`): *point* in origin of $\{o^{1a}\}$.
- $\widehat{\boldsymbol{ha}}_{\{o^{1a}\}}$ (`handle_axis_direction`): *versor* aligned with the $z-$axis of $\{o^{1a}\}$.
- $\overline{\boldsymbol{hx}}_{\{o^{1a}\}}$, $\overline{\boldsymbol{hy}}_{\{o^{1a}\}}$, $\overline{\boldsymbol{hz}}_{\{o^{1a}\}}$ `handle_axis_(x-y-z)`: three *lines* with origin in $\{o^{1a}\}$, and aligned with the $x-$, $y-$, or $z-$axis, respectively.
- $\overline{\boldsymbol{oa}}_{\{o^{1b}\}}$ `opening_axis`: *line* with origin in $\{o^{1b}\}$, and aligned with the opening direction.
- $\boldsymbol{gp}_{\{o^{2}\}}$ (`grasp_position`): *point* in origin of $\{o^{2}\}$.
- $\widehat{\boldsymbol{gn}}_{\{o^{2}\}}$ (`grasp_normal_direction`): *versor* aligned with the $x-$axis of $\{o^{2}\}$.
- $\overline{\boldsymbol{ga}}_{\{o^{2}\}}$ `grasping_axis`: *line* with origin in $\{o^{2}\}$, and aligned with its $z-$axis.

B. "Open the drawer" tasks description

The example application is ruled by a simple three-states machine, whose state transitions are executed in response to the event raised by monitors (described in Sec. VII-C). The states (*approach the tray*, *grasp the handle*, and *open the drawer*) are ordered sequentially, and, for each state, a different set of *task* is enforced. In each task a set of *primary constraint* is enforced; in addition, in each task, a *safety constraint* is enforced in order to limit the joint positions in their range, (*Position Limit behaviour*).

a) Approach the tray (S.1): In this phase, the robot should bring its end effector in such a position that can conveniently grasps the handle. To do so, the robotic hand should be: *i)* oriented toward the object, *ii)* rotated along its z- (approach-) axis in the "correct" way, and then *iii)* bring the distance between the grasping point and the handle center to zero. This description translates in a set of Positioning behaviours, ruled by the following expressions:

a) *line-point distance* between $\overline{\boldsymbol{ga}}_{\{o^{2}\}}$ and $\boldsymbol{hp}_{\{o^{1a}\}}$ should be zero,
b) *angle btw versors* $\widehat{\boldsymbol{gn}}_{\{o^{2}\}}$ and $\widehat{\boldsymbol{ha}}_{\{o^{1a}\}}$ should be zero,
c) while three *line-point projections* $\left(\overline{\boldsymbol{hx}}_{\{o^{1a}\}}, \boldsymbol{gp}_{\{o^{2}\}}\right)$, $\left(\overline{\boldsymbol{hy}}_{\{o^{1a}\}}, \boldsymbol{gp}_{\{o^{2}\}}\right)$, and $\left(\overline{\boldsymbol{hz}}_{\{o^{1a}\}}, \boldsymbol{gp}_{\{o^{2}\}}\right)$, should go to zero.

These set of constraints allow for a one full degree of freedom (angle of approach to the handle around its normal axis). Note that expressions c) dictate that the distance between the origins of the frames $\{o^{1a}\}$ and $\{o^{2}\}$ be zero. However, we express three constraints in place of a *point-point distance* as the derivative of such expression is ill-defined at the desired value, thus causing local instability around such point.

b) Grasp the handle (S.2): While the positioning constraints are held, we close the gripper.

c) Open the drawer (S.3): At this point, the drawer is opened, so we command to increase the distance along the line direction of opening, by increasing the point-line projection between $\overline{\boldsymbol{oa}}_{\{o^{2}\}}$ and $\boldsymbol{gp}_{\{o^{2}\}}$. As we do not want the handle grasp to be broken, we continue to enforce previous constrains.

C. Monitors and trajectory generator

For this example, we limit a sketch of possible monitors that can dictate the success of the action, leaving out the management of other occurrences.

a) Approach the tray: the final goal is to reach the center of the handle. We use the *point-point distance* between the origins of frames $\{o^{1a}\}$ and $\{o^{2}\}$. In this case, the monitored expression is different to the controlled ones (the three *line-point projections*).
b) Grasp the handle: success can be achieved by force direct/indirect measurement in the grasp space. However, we rely on position information (the distance between fingertips).
c) Open the drawer: success can be acknowledged by monitoring when the distance between the gripper position and the initial drawer position is above a given threshold.

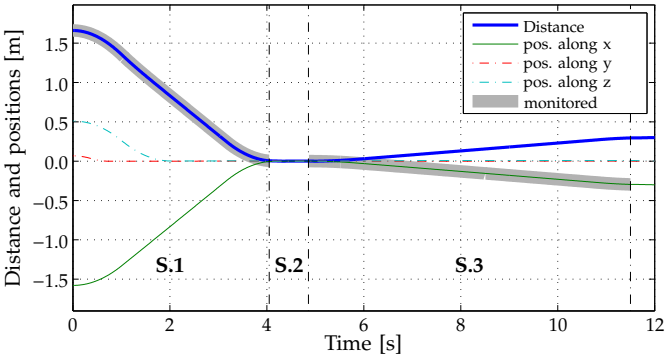
All the constraints that are described are commanded to the final value by means of trapezoidal trajectory generator. The results can be appreciated in Fig. 4, where the time evolution of all expressions are reported. In these figures we reported also which is the monitored expression, and the transitions between tasks.

VIII. Conclusion

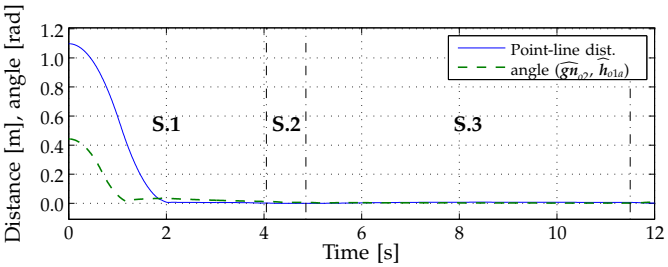
This paper formalises task representations, using a strictly specified set of symbolic elements (and very few numerical values). The aim is to help the *unification* of the various state-of-the-art approaches that deal with complex tasks on complex robotic systems. The benefit of this process is manifold: by defining constraints with a limited number of formally described elements, it becomes possible to design *reasoning* algorithms to sort out the best plan for a given action (*e.g.* [11], [13]) and check (semantically) whether a task is ill- or well-defined, without prior knowledge of the solver specifics, and/or with a lot less involvement of human developers. This progress helps in the "ease of use" of task specification, but also in "verification & validation" of robotic applications. The set of primitives and relations can be extended, both for the geometric expressions (*e.g.* segments, planar polygons, solids, oriented curves to provide virtual guidance, ...), or any other expressions that meets the above-mentioned criteria.

References

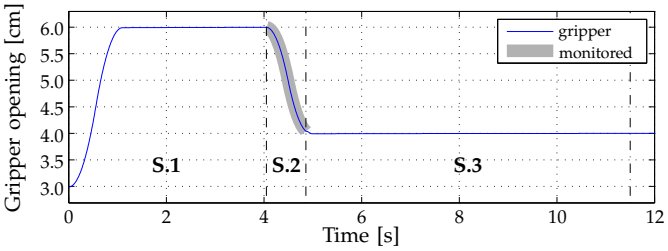
[1] M. T. Mason, "Compliance and Force Control for Computer Controlled Manipulators," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 11, no. 6, pp. 418–432, 1981.
[2] N. Mansard, O. Khatib, and A. Kheddar, "A Unified Approach to Integrate Unilateral Constraints in the Stack of Tasks," *IEEE Transactions on Robotics*, vol. 25, no. 3, pp. 670–685, 2009.
[3] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decre, R. Smits, E. Aertbelin, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *International Journal of Robotic Research*, vol. 26, no. 5, pp. 433–455, 2007.



(a) x, y, z positions of gripper w.r.t. the handle (initial, fixed) frame, and total distance between frame origins.



(b) Point-line distance between the gripper grasping direction and the handle frame origin, and misalignment between the handle axis and normal grasping directions.



(c) Distance between gripper fingers.

Figure 4. Values of constrained and monitored expressions during simulation. Vertical lines shows the transitions between states (labelled from **S.1** to **S.3**, each one corresponding to a set of *primary constraints*). In the first state, the gripper is opened (Fig. 4c), aligned (Fig. 4b), and brought to the handler (Fig. 4a). State transitions is triggered when total distance (blue flat line in Fig. 4a) decreases under a given threshold. During **S.2** the gripper is closed (Fig. 4c), and, lastly, the gripper is commanded to move back to $x = -0.3\,\mathrm{m}$ (thin solid line in Fig. 4a). The monitored expressions in each task are highlighted with a fat gray underling line.

[4] L. Sentis and O. Khatib, "A Whole-Body Control Framework for Humanoids Operating in Human Environments," in *IEEE Proc. of the Int. Conf. on Robotics and Automation*, 2006.

[5] H. Bruyninckx and J. De Schutter, "Specification of force-controlled actions in the "task frame formalism" - a synthesis," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 581–589, 1996.

[6] T. Kröger, B. Finkemeyer, and F. M. Wahl, "Manipulation Primitives - A Universal Interface between Sensor-Based Motion Control and Robot Programming," in *Robotic Systems for Handling and Assembly*, ser. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, 2011, vol. 67, pp. 293–313.

[7] D. Vanthienen, M. Klotzbucher, and H. Bruyninckx, "The 5C-based architectural Composition Pattern: lessons learned from re-developing the iTaSC framework for constraint-based robot programming," *Journal of Software Engineering for Robotics*, vol. 5, no. 1, 2014.

[8] T. Kröger, B. Finkemeyer, and F. M. Wahl, "A Task Frame Formalism for Practical Implementations," in *IEEE International Conference on Robotics and Automation*, New Orleans, LA, USA, April 2004, pp. 5218–5223.

[9] T. Bray, "RFC 7159, The JavaScript Object Notation (JSON) Data Interchange Format," August 2014. [Online]. Available: https://tools.ietf.org/html/rfc7159

[10] F. Galiegue and K. Zyp, "JSON Schema, Draft 0.4v," August 2013. [Online]. Available: http://tools.ietf.org/html/draft-zyp-json-schema-04

[11] M. Beetz, L. Mosenlechner, and M. Tenorth, "CRAM– A Cognitive Robot Abstract Machine for everyday manipulation in human environments," in *IEEE/RSJ Proc. of International Conference on Intelligent Robots*, 2010, pp. 1012–1017.

[12] P. Doherty and F. Heintz, "A delegation-based cooperative robotic framework," in *IEEE International Conference on Robotics and Biomimetics*, Dec 2011, pp. 2955–2962.

[13] E. Scioni, G. Borghesan, H. Bruyninckx, and M. Bonfe, "Bridging the gap between discrete symbolic planning and optimization-based robot control," in *IEEE International Conference on Robotics and Automation*, May 2015, pp. 5075–5081.

[14] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation," in *IEEE International Conference on Robotics and Automation*, May 2011, pp. 4575–4581.

[15] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *International Journal of Robotics Research*, vol. 32, no. 9-10, 2013.

[16] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *International Journal of Robotic Research*, vol. 33, no. 14, pp. 1726–1747, 2014.

[17] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *International Journal of Robotics Research (IJRR)*, vol. 30, no. 12, pp. 1435 – 1460, October 2011.

[18] L. B. Rosenberg, "The use of virtual fixtures as perceptual overlays to enhance operator performance in remote environments," Wright Patterson Air Force Base, OH: U.S.A.F Armstrong Laboratory, Technical Report AL-TR-1992-XXX, 1992.

[19] C. Sayers, *Remote control robotics*. Springer Verlag, 1999.

[20] J. Abbott, P. Marayong, and A. Okamura, "Haptic Virtual Fixtures for Robot-Assisted Manipulation," in *Robotics Research*, ser. Springer Tracts in Advanced Robotics, 2007, pp. 49–64.

[21] L. D. Joly and C. Andriot, "Imposing motion constraints to a force reflecting telerobot through real-time simulation of a virtual mechanism," in *IEEE International Conference on Robotics and Automation*, Nagoya, Japan, 21-27 May 1995, pp. 357–362.

[22] K. Kosuge, T. Itoh, T. Fukuda, and M. Otsuka, "Tele-manipulation system based on task-oriented virtual tool," in *IEEE International Conference on Robotics and Automation*, vol. 1, Nagoya, Japan, 21-27 May 1995, pp. 351–356.

[23] E. Scioni, G. Borghesan, H. Bruyninckx, and M. Bonfe, "A framework for formal specification of robotic constraint-based tasks and their concurrent execution with online qos monitoring," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2014, pp. 2963–2969.

[24] G. Borghesan, E. Aertbeliën, and J. De Schutter, "Constraint- and synergy-based specification of manipulation tasks," in *IEEE Proc. of the Int. Conf. on Robotics and Automation*, 2014.

[25] E. Aertbeliën and J. De Schutter, "eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs," in *IEEE/RSJ International Conference on Intelligent Robots*, 2014.